(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 021 365**
A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 80103457.0

(22) Date of filing: 20.06.80

(51) Int. Cl.³: G 06 F 15/16, G 06 F 9/38, G 06 F 15/20

(30) Priority: 28.06.79 JP 79682/79

(43) Date of publication of application: 07.01.81 Bulletin 81/1

(84) Designated Contracting States: DE FR GB NL

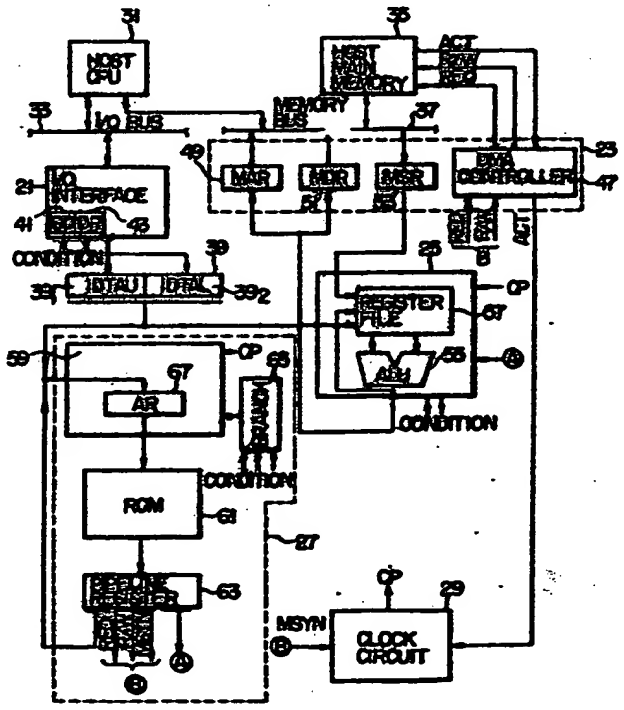
(71) Applicant: TOKYO SHIBAURA DENKI KABUSHIKI KAISHA, 72, Horikawa-cho Saiwai-ku, Kawasaki-shi Kanagawa-ken 210 (JP)

(72) Inventor: Shibayama, Shigeki, 1020 Hiyoshihon-cho Kohoku-ku, Yokohama-shi (JP)
Inventor: Iwata, Kazuhide, 691-17-9-201, Kami Shirano-cho Asahi-Ku, Yokohama-shi (JP)

(74) Representative: Patentanwälte Henkel-Kern-Feiler-Hänzel, Möhlstrasse 37, D-8000 München 80 (DE)

(54) Data processing system with a slave computer.

(57) In a data processing system having a slave computer connecting to a host central processing unit (31) and a host main memory (35), the slave computer includes an arithmetic logic operating means (25). The arithmetic logic operating means (25) calculates an address of the host main memory (35). A DMA interface (23) directly makes an access to the host main memory (35) to fetch operand data into the slave computer. The arithmetic logic operating means (25) computes the operand data under control of a microprogram control section (27) and directly loads the computed result to the host main memory (35) through the DMA interface (23).



EP 0 021 365 A2

Data processing system with a slave computer

The present invention relates to an electronic computer system and more particularly to a computer system which is the combination of an ordinary computer system and an auxiliary computer system whereby the process of which the execution by the ordinary computer is made merely at a low speed is carried out by the auxiliary computer additionally used and therefore the computer system operates at a high speed.

By convention, the computation as an excessive task for the host computer is frequently assigned to an additionally used computer system, or a slave processor, appropriate for the computation. Such a slave computer system is commercially available now. A typical example of such is an array processor which executes the computation of array type data at high speed, such as an array processor (AP 120B) by Floating Point Systems Inc. in USA and AP 400 by Analogic Inc. in USA. The detail of the AP 400 is discussed in US Patent No. 4,135,247 granted to Gordon et al on January 16, 1979. Each of those systems has a high speed memory and an arithmetic logic unit. In operation, the computation data is fetched from a main memory of a host computer and the fetched data is loaded into the high speed memory and then is computed by an arithmetic logic unit. The computed data is loaded into the high speed memory and

afterwards is transferred to the main memory. In this way, a sequence of the data processing operation is completed. This type apparatus is effective for an operation having many intermediate results of computation such as a First Fourier Transform but is improper for a relatively simple processing in which the data in the main memory is subjected to a simple computation with little production of the intermediate computation result. The reason for this is that if the high speed memory holds the intermediate computation results and the results are accessed many times, the high speed feature is effectively used; however, with respect to the processing requiring no intermediate computation results, it operates merely as a buffer between the main memory and the arithmetic logic unit. Thus, it is ineffective in utilizing the high speed feature thereof.

When $c_i = a_i + b_i$, $i = 1, 1000$ is computed by using the conventional array processor (a_i and b_i are stored in the main memory and the memory location for storing c_i is also in the main memory), the data a_i and b_i are read out from the main memory and are transferred to an internal memory. Then, the addition for obtaining c_i is performed by an internal arithmetic logic unit. The result of the addition is stored in the internal memory. After all of the results of the additions are computed, the total result is returned from the internal memory to the main memory. This operation is diagrammatically illustrated in the time scale in Fig. 1.

As a first step, the array data a_i is read out from an address, which is generally needed at the time of the set-up and given in some way, and is transferred to the memory of the slave processor. This step is performed in a section α. The necessary address computation in this case is performed by the slave processor. In the next step, the array data b_i is likewise fetched from

the main memory. This is performed in a section β. When the fetching of all of the array data a_i and b_i is completed, the slave processor reads out the array data a_i and b_i from the internal memory, adds those and loads again the result c_i into the internal memory by using the internal arithmetic logic unit. Normally, the internal memory and the arithmetic logic unit of the slave computer are higher than the host computer, and thus processing in this section is at high speed. This operation is performed in a section γ. When the computation (the addition in this case) of all of the array data are completed, the result of c_i is transferred (loaded) from the internal memory to the main memory of the host computer. This operation is performed in a section γ. Assuming that an access time of the main memory is τ_{am}, and a processing time of the arithmetic logic unit is τ_{pr}, the total of the computation time (provided that the main memory could be accessed exclusively), is given

$$1000\tau_{am}+1000\tau_{am}+1000\tau_{pr}+1000\tau_{am} = 3000\tau_{am}+1000\tau_{pr}$$

Therefore, even if the arithmetic logic unit is operable at high speed ($\tau_{pr}\to 0$), the arithmetic time is determined by the access time $3000\tau_{am}$ to the main memory. Accordingly, it is undesirable that the internal high speed memory is particularly provided in the slave computer for effecting such a processing. This is for the reason that the high speed performance little contributes to the reduction of the processing time and that the high speed internal memory is expensive.

Accordingly, an object of the present invention is to provide a computer system with a good performance cost.

Another object of the present invention is to provide a computer system with a slave computer having no internal memory.

To achieve the above objects, there is provided a

data processing system with a slave computer connecting to a host central processing unit and a host main memory characterized in that the slave computer comprising: arithmetic logic operating means of which the operation cycle is higher than the memory cycle of the host main memory, and computes the address of the host main memory; DMA interface means which is connected to the host main memory and the arithmetic logic operating means, directly makes an access to the host main memory in accordance with the address of the host main memory computed by the arithmetic logic operating means, and directly store the operation result of the arithmetic logic operating means in the host main memory; microprogram control means connected to the arithmetic logic operating means and the DMA interface means for controlling those means; input/output interface means connected to the host central processing unit and the microprogram control means to apply command information received from the host central processing unit to the microprogram control means; and clock circuit means connected to the arithmetic logic operating means, the microprogram control means and the DMA interface means for generating system clocks to operate those means.

In the present invention, a slave computer system with no internal memory, which is expensive, used for the array data computation, is connected to a host computer. The slave computer computes the data obtained by directly making an access to the main memory of the host computer system through an interface, and loads the result of the computation to the main memory through the interface, again.

In a preferred embodiment of the present invention, the slave computer has an arithmetic logic unit with a processing speed several times the memory cycle of the main memory, which the arithmetic logic unit is controlled by a microprogram control unit. The

microprogram control unit has a number of microprogram routines corresponding to the computations. The host computer system drives the slave computer system to perform the computation having an extremely long time when it is made by the host CPU. Specifically, the host CPU gives a microprogram routine to be executed and information to indicate where operand data is stored in the main memory as well, to the slave computer. Afterwards it gives a computation start command signal to the slave computer, so that the slave computer starts the processing related. Within one memory cycle, the arithmetic logic unit executes a plurality of steps of the processing including the address computations of the data to be read from or written in the main memory, as well as the normal computation. A great amount of data in the main memory is accordingly processed in a pipeline manner. For the pipeline processing, the slave computer must be synchronized with the main memory. To this end, the slave computer has an internal clock generator and means for inhibiting the oscillation of the internal clock for synchronizing with the memory cycle of the main memory.

Other objects and features of the invention will be apparent from the following description taken in connection with the accompanying drawings, in which:

Fig. 1 is a time scale useful in explaining the computation of the array data by the conventional technique;

Fig. 2 is a block diagram of an embodiment of an electronic computer system according to the invention;

Fig. 3 is a block diagram of a detailed DMA controller shown in Fig. 2;

Fig. 4 is a memory map of a host main memory shown in Fig. 2;

Figs. 5A and 5B cooperate to form a flow chart of a set-up sequence for computing the data array, which is

useful in explaining the operation of an embodiment of the computer system according to the invention;

Fig. 6 is a memory map of the microprogram shown in Fig. 2;

Fig. 7 is a block diagram of a detailed branching condition circuit shown in Fig. 2;

Figs. 8A and 8B cooperate to form a flow chart of a computing sequence of the array data;

Fig. 9 is a format of a microinstruction used in the embodiment;

Fig. 10 is a timing chart illustrating timings of the operation of the embodiment shown in Fig. 2;

Fig. 11 is a block diagram of a clock circuit shown in Fig. 2; and

Fig. 12 graphically illustrates relations of addresses and mnemonic contents.

Reference is made to Fig. 2 illustrating a construction of a slave computer comprising an I/O interface 21, a DMA interface 23, an arithmetic logic section 25, a microprogram control unit 27 and a clock circuit 29. The slave computer is connected through an I/O bus 33 to a host CPU 31 and through a memory bus 37 to a host main memory 35. A host computer system used in the present embodiment is a conventional ordinary mini-computer.

The I/O interface 21 provides an interface between the slave computer and the host computer through the I/O bus 33 and gives a microinstruction (start address) to the slave computer and an interruption to the host CPU 31.

The data fed from the host CPU 31 is stored in an IDTA register 39. The I/O interface 21 has a flip-flop GO 41 and a flip-flop DR 43 for indicating a state of the I/O interface 21.

A DMA interface 23 fetches computation data from a main memory 35 or loads the result of computation by the

arithmetic logic section 25 to the main memory 35, which the interface 23 has a DMA controller 47, and various registers such as MAR 49, MDR 51 and MSR 53. The MAR 49 holds an address to fetch data from the main memory 35 or an address to store data into the main memory 35. The MDR 51 holds the computation result to be stored into the main memory 35. The MSR 53 holds the computation data fetched from the main memory 35.

The DMA controller 47 is comprised of an RS flip-flop 48 and a D flip-flop 50, as shown in Fig. 3. The RS flip-flop 48 holds a memory request signal for the host main memory 35. The D flip-flop 50 holds a read/write signal for the host main memory 35. The RS flip-flop 48 is set by the memory request signal and reset by an acknowledge signal derived from the host main memory 35. The D flip-flop 50 is set by the acknowledge signal. The acknowledge signal, when the memory request signal and the read/write signal are supplied to the host main memory 35, is an answer back signal outputted from the host main memory 35 to the DMA controller 47. The memory request signal and the read/write signal are produced from a pipeline register 63 of a microprogram control unit 27 to be described later. The DMA controller 47 bypasses the acknowledge signal to a clock circuit 29 to be described later.

An arithmetic logic section 25 has an ALU 55 and a register file 57 which executes the computation directed by a microprogram control section 27. The embodiment uses four 4-bit slice bipolar microprocessors (Am 2901) manufactured by AMD Co., USA, an ALU of 16 bits and 16 registers.

A microprogram control section 27 comprises a microprogram sequencer 59, a microprogram memory (ROM) 61, a pipeline register 63 and a branch condition circuit 65.

A microprogram sequencer 59 such as a microprogram sequencer (Am 2910) manufactured by AMD Co., USA, has an

address register 67 therein. The microprogram memory (ROM) 61 holds an operation routine performed by the arithmetic logic section 25 and a set-up routine to be described later. By an address held by the address register 67, a specific microinstruction in a specific routine is read out from a microprogram memory (ROM) 61 and is loaded into a pipeline register 63. The arithmetic logic section 25, a DMA interface 23 and the various registers are controlled in accordance with the microinstruction. The branch condition circuit 65 provides a branch command to the microprogram sequencer 59 to branch the microprogram in accordance with various flip-flop states within the arithmetic logic section 25 and the I/O interface 21.

The clock circuit 29 has a function as a synchronizing circuit with respect to the memory access, as described later, as well as a function to send a clock signal CP to various sections of the slave computer.

The operation of the computer system as mentioned above will be described in detail. As mapped in Fig. 4, the main memory 35 stores 1000 data a_i (i = 1, 1000) in addresses 1000 to 1999, 1000 data b_i (i = 1, 1000) in addresses 2000 to 2999, and the addition result c_i (i = 1, 1000) of those corresponding data in addresses 3000 to 3999.

Firstly, the host CPU 31 provides a set-up operation. In Figs. 5A and 5B, a flow chart of a set-up sequence is illustrated.

The set-up sequence starts in response to an initialize command delivered through the I/O interface 21 from the host CPU 31. The initialize command initializes the various flip-flops (including those not shown) within the slave computer and the microprograms as well. In the ROM 61, the set-up routine and various arithmetic routines are stored as shown in Fig. 6. In response to the initialize command, the set-up routine

starts. Then, the host CPU 31 transfers a code representing $c_i = a_i + b_i$ to the IDTA register 39. Since the IDTA register 39 has a 16-bit width, the code is transferred two times and is set in the registers IDTAU 39_1 and IDTAL 39_2 having each an 8-bit width.

In a block 81, the I/O interface 21 checks if the data (8-bit data of 16 bits) delivered from the host CPU 31 is present or not and, if not present, holds the wait loop till the data appears. If the data is present, the I/O interface 21 loads the data into the IDTAU register 39_1 (block 83). Then, in a block 85, the I/O interface 21 checks if the next data (the remaining 8-bit data of the 16 bits) is present or not. If it is absent, the interface 21 holds a wait loop till the data appears. If the data is present, the interface 21 loads the data into the IDTAL register 39_2 (block 87). The code of the IDTA register 39 thus obtained is a start address of the arithmetic routine denoted as #2 for computing $c_i = a_i + b_i$. When data is set in the IDTA register 39, the DR flip-flop 43 is set and the set-up routine loads the contents of the IDTA register 39 into the address register AR 67, when the DR flip-flop 43 detects the set (block 89).

Then, the I/O interface 21 checks if the host CPU 31 has transferred the table address (8 bits of the 16 bits) or not. If the address is transferred from the host CPU 31, the I/O interface 21 receives and loads it into the IDTAU register 39_1 (block 93). The I/O interface 21 receives the table address of the remaining 8 bits from the host CPU 31 and loads it into the IDTAL register 39_2 (blocks 95 and 99). The table address indicates location in which a table representing the first addresses of the data a_i, b_i and c_i (addresses 1000, 2000 and 3000) and the number (1000) of the data, is stored in the main memory 35.

As shown in Fig. 4, if the table is stored in the

addresses 500 to 503, the data "500" is loaded as a table address into the IDTA register 39. The data in the IDTA register 39 is defined by the table handled in the microprogram, so that the contents of the table changes as the kind of the operation to be executed changes.

Through the above-mentioned operation, the data necessary to start the operation are set up and therefore the host CPU 31 sends a GO command to the I/O interface 21. The GO flip-flop 41 is set by the GO command. A state of the GO flip-flop 41 has been supplied to the branch condition circuit 65. As shown in a block 99, when the set-up routine detects this, the microprogram is branched to an address held in the address register 67 and the execution of the arithmetic routine #2 starts (block 101).

The branching condition circuit 65 will briefly be described referring to Fig. 7. In the figure, the branch condition circuit 65 is comprised of a condition code selector 69 and an polarity control exclusive OR 71. The selector 69, which has been supplied with the information of the flip-flops, selects the contents of any of the flip-flops. Upon the selected information, the exclusive OR 71 supplies a signal to direct the branch to the microprogram sequencer 59. In this case, if the arithmetic routine #2 is allocated in the address 100 of the ROM 61, "100" set in the pipeline register 63 is supplied to the ROM 61 through a selector (not shown) and a branch to the address 100 takes place. Of course, the address 100 is related to the microprogram and not to the main memory. The arithmetic routine #2 starting at the address 100 is performed as shown in Fig. 8, for example. The central portion of such a program, or an addition loop, is of the usual type. In a block 103, the contents of the IDTA register 39 is loaded into the arithmetic logic section 25. The first address of a table necessary for starting the

operation as previously stated has been prestored in the IDTA register 39. In this example, an address "500" is loaded into the arithmetic logic section 25. The arithmetic logic section 25 transfers the address "500" to the MAR 49 to give a Read command to the main memory 35 through the DMA interface 23. Here, "Read" indicates a direction for the slave computer to read (data flows main memory to slave processor). In response to the read command, the DMA interface 23 reads the contents of the address 500 of the main memory 35 (1000 of the start address of the array a_i, in this example,) and sets the read one into the MSR 53 (block 105). The arithmetic logic section 25 loads the first address of the array a_i, or the contents of the MSR 53, into the internal register. Similarly, while incrementing the address, the first address of the b_i array, the first address of the c_i array, and the number of the operations are sequentially loaded into the register of the arithmetic logic section 25 (blocks 107, 109 and 111). Then in blocks 115 and 117, a_k data ($k = 1, 1000$) and b_k data ($k = 1, 1000$) are fetched and, in a block 119, a_k data and b_k data are added and the result of the addition c_k is stored into c_k. Then, in a block 121 the c_k is stored into the memory.

Then, in a block 123 a pointer K is incremented by one. In a block 125 the pointer K is compared with the number of data N. In the comparison, when the value of the pointer K fails to reach the N, the control is returned to the block 115 where similar process and computation are performed. The operation loop of the blocks 115 to 125 performs an ordinary operation in the flow chart shown in Fig. 8. In the slave computer, however, the fetch and the addition of the data are overlapped in a pipeline manner and all of the operations are completed within the access time of the main memory 35.

An example of the addition program is illustrated in TABLE 1. The operation of the arithmetic logic section 25 and the memory request are described in the microprogram of the ROM 61. The basic cycle of the slave computer is 200 ns, for example, and the cycle time of the main memory is 800 ns, for example. Therefore, the slave computer can perform one step of the program for one memory cycle.

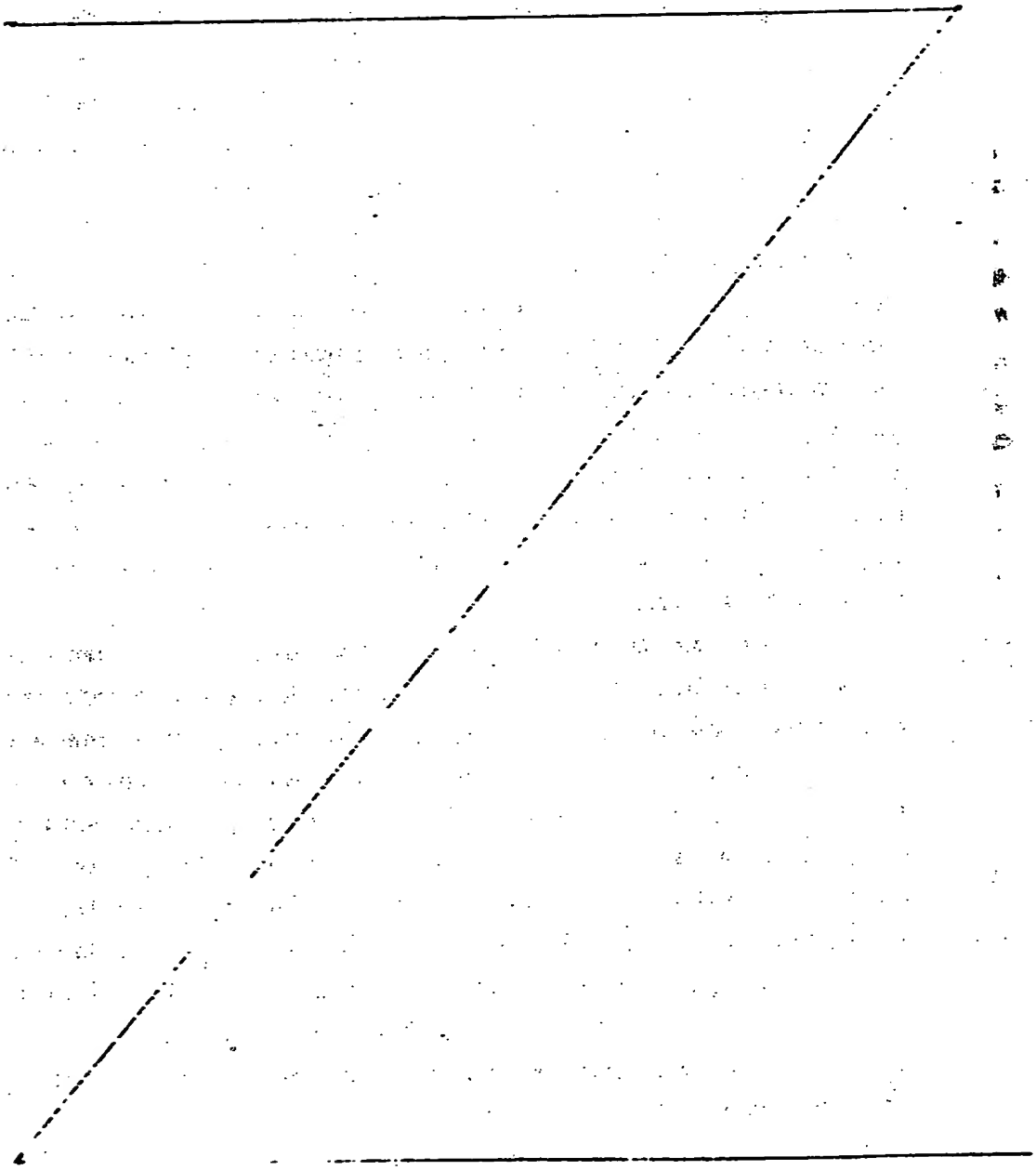


TABLE 1

ADDRESS	ARITHMETIC SECTION	MEMORY REQUEST/ SYNCHRONIZE	REMARKS
a	k←0	REQ(R)	COUNT CLEAR, READ REQUEST
a+1	AA→MAR	MSYN	SEND a_1 ADDRESS
a+2	AA incr		UPDATE a ARRAY ADDRESS
a+3	nop		
a+4		REQ(R)	READ REQUEST
a+5	BA→MAR	MSYN	SEND b_1 ADDRESS
a+6	MAR→R_1		TRANSFER a_1 DATA TO R_1
a+7	BA incr		UPDATE b ARRAY ADDRESS
a+8	nop	REQ(R)	READ REQUEST
a+9	AA→MAR	MSYN	SEND a_i ADDRESS
a+10	MSR+R_1→R_2		$b_1+a_1 \rightarrow R_2(c_1)$
a+11	AA incr		UPDATE a ARRAY ADDRESS
a+12	nop	REQ(R)	READ REQUEST
a+13	→LOOP BA→MAR	MSYN	SEND b_i ADDRESS
a+14	MSR→R_1		TRANSFER a_i DATA TO R_1
a+15	BA incr		UPDATE b ARRAY ADDRESS
a+16	R_2→MDR	REQ(W)	SEND c_{i-1} TO MDR
a+17	CA→MAR	MSYN	SEND c_{i-1} ADDRESS
a+18	MSR+R_1→R_2		$b_i+a_i \rightarrow R_2(c_i)$
a+19	CA incr		UPDATE c ARRAY ADDRESS
a+20	k incr	REQ(R)	COUNT UP
a+21	AA→MAR	MSYN	SEND a_{i+1} ADDRESS
a+22	AA incr		UPDATE a ARRAY ADDRESS
a+23	k-N		} CHECK THE NUMBER OF TIMES
a+24	BNZ LOOP		
a+25	STOP		END

Each microinstruction of the arithmetic routine has a memory control section and an arithmetic control section, as shown in Fig. 9. In TABLE 1, the arithmetic section indicates the operations described in the arithmetic control section and the memory request/synchronize indicates the operations described in the table. The arithmetic control section has an OP code, a source address, and a destination address, which are usually used. The description of those will be omitted, however. The memory control section has three bits REQ, R/W and MSYN, as illustrated enlarged in Fig. 9. The REQ bit is a bit to direct the request to the main memory 35, the R/W bit is a bit to direct if the request is the read or the write for the main memory 35, and the MSYN bit indicates the synchronization with the memory cycle. Of those bits, the contents of the REQ bit and R/W bit are supplied to the DMA controller 47 and the contents of the MSYN bit is supplied to the clock circuit 29. The DMA controller 47 produces a request signal and an R/W signal to the main memory 35 in accordance with the contents of the REQ bit and the R/W bit.

In TABLE 1, REQ(R) indicates that the REQ bit and the R/W bit are set (to Read). Similarly, REQ(W) indicates that REQ bit and R/W bit are set (to Write).

Upon the memory request, the memory cycle of the main memory 35 starts. In this example, when the memory request with higher priority is received, the memory cycle is so programmed as to start delayed by two steps from the step which issued the memory request. In other words, after approximately 400 ns since the memory request is issued, an actual memory cycle starts corresponding to the memory request.

In TABLE 1, symbols k, AA, BA, N, R_1, R_2 and CA are names assigned to specific registers in the register file 57. In those names, k designates counters

corresponding to the number of operations; AA an address of the array data a; BA an address of the array data b; CA an address of the array data c; N the number of operations (the number of data) previously given by the table; R_1 and R_2 working registers. As the present embodiment uses the Am 2901 for the arithmetic section 25, the MAR 49 and MDR 51 are connected to the Y output port of the AM 2901, and the MSR 53 is connected to the D input port. Accordingly, the loading and the fetching operations of the contents of these registers are possible by the microprogram.

In the microprogram in TABLE 1, the addresses from a to a+12 indicate an initializing step since it is impossible to process those addresses from the first data in the pipeline manner. The steps from the address (a+13) to the address (a+24) indicates a loop in a normal state.

In the cycle of the main memory 35, a_i and b_i (a and b represent "Read") are read such as (a_1, b_1), (a_2, b_2, c_1), (a_3, b_3, c_2), (a_4, b_4, c_3) ... and c_{i-1} obtained in the previous cycle is written. The process of four internal cycles is possible within one memory cycle. Accordingly, the address control of DMA and the count of the loop, and the like are overlapped with the memory cycle, as shown in the program, so that the computation is pipelined and all of the computations are completed within the access time to the memory.

The microprogram in TABLE 1 will again be described with relation to the operation of the host main memory 35. One memory cycle of the host main memory 35 corresponds to four steps of the microprogram. Accordingly, with MSYN of the address (a+1), the step of the address (a+2) is synchronized with the memory cycle. Thus, the memory cycle starts after two steps since the memory request is issued. Therefore, the time taken from the address (a+2) to the address (a+5) correspond

to one memory cycle and here the array data a_1 is Read. Likewise, the addresses (a+b) to (a+9) correspond to one memory cycle and here the array data b_1 is Read. Similarly, the addresses (a+10) to (a+13) are for Read of a_2; the addresses (a+14) to (a+17) for Read of b_i; the addresses (a+18) to (a+21) for Write of c_{i-1}; the addresses (a+22) to (a+24) for Read of a_{i+1}.

In the program, when the access request to the host main memory 35 is issued, the memory cycle starts immediately (after approximately 400 ns). The actual host main memory 35, however, is commonly used by the CPU 31, the slave computer and DMA devices such as a magnetic disc and a magnetic tape. The devices individually request services for the host main memory 35 and exclusively use the memory cycle only when the request is accepted. Also during a period that the program shown in this example runs, it is impossible for the slave computer to continuously use the memory cycle. In this case, it must operate using the memory with any other device. To this end, the synchronizing circuit is provided in the slave computer so that from an instant that the access request to the host main memory 35 is issued till an acknowledge signal representing the request is accepted is received, the internal clock is stopped and the slave computer is operated always to synchronize with the memory cycle of the main memory assigned thereto. The operation timing diagram for taking such a synchronization is illustrated in Figs. 10A to 10J. An example of the circuit construction of the clock circuit 29 which also serves as a synchronizing circuit is shown in Fig. 11. MCLK in Fig. 10B designates a clock signal generated by the oscillating circuit 81. The clock signal is frequency-divided by the flip-flops 83 and 85 thereby to form clock signals SCLK1 and SCLK2. These signals SCLK1, SCLK2 are supplied to a disable flip-flop 87

through a logic multiplying circuit 88. The machine cycle is determined by the clock signal SCLK1.

The memory synchronizing operation is performed by the microprogram as shown in Fig. 12, for example. That is at address b, an access request to the memory is issued and the machine cycle at the address b+1 should be elongated till the memory cycle is assigned to synchronize therewith. Accordingly, the memory synchronizing operation is specified (M, SYNC command). The execution of the program will be described using the timing chart. When the program in the address b+1 is executed after the program in the address b is executed, the M and SYNC commands cause the MSYN signal to synchronize with the SCLK1 thereby to be active. The DISA flip-flop 87 is set by the timing pulse TP formed by the SCLK1 and SCLK2, so that the output $\overline{Q}$ of the DISA flip-flop 54 resets the flip-flops 83 and 85. Under this condition, the clock signals SCLK1 and SCLK2 are halted. Then, the main memory returns a signal ACT to accept the access to the memory, to the circuit. At the leading edge of the ACT signal, the ACT latch 89 is set. Since the ACT signal is of asynchronous, the ACT latch 89 is made to be synchronized with the MCLK and the DISA flip-flop 87 is reset by the output $\overline{Q}$ of the SYNC flip-flop 91. Upon the reset, the output $\overline{Q}$ of the DISA flip-flop 87, which have reset the flip-flops 83 and 85 becomes inactive, the SCLK1 and SCLK2 become active again. At this time point, the execution of the program starts to progress and the execution of the program in the address b+2 is started. By performing such synchronizing operation, the internal operation may be performed with respect to the common use of the memory, correctly and smoothly.

According to the invention, the slave computer has no internal memory to hold a large amount of data. Accordingly, a large amount of data may be processed at

a high speed by using an inexpensive slave computer, so that the computer system of the invention has a high performance/cost ratio. In the above-mentioned embodiment, a space to occupy the slave computer on a single print substrate of 12 × 12 inches is almost same compared with the conventional one. The additional use of the slave computer never interfere the host computer system but takes charge of processing the picture data, and the array data, which are not well processed by the host computer thereby to improve the data processing speed of the whole system. The operation for the host computer to start the slave computer is simplified and this indicates that the computer system is easy to use from the host computer side.

A preferable application of the invention is a computed tomography (CT) which irradiates the X ray beam onto the exterior of a patient to collect the data transmitted therethrough and to obtain a tomogram of the diseased part of the patient by processing the transmitted data by means of a computer. To obtain the tomogram, a calculation called a back projection is performed. The amount of the data used in the back projection ranges from several tens thousand to several hundreds thousand. Such an enormous amount of data may be processed for a short time by the computer system of the present invention. The CT apparatus includes a computer system generally and has an extremely expensive. The use of the computer system of the invention reduces the cost of the CT apparatus.

Claims:

1. In a data processing system with a slave computer connecting to a host central processing unit and a host main memory, the improvement wherein said slave computer comprising:

arithmetic logic operating means of which the operation cycle time is shorter than the memory cycle time of said host main memory, and computes the address of the host main memory;

DMA interface means which is connected to said host main memory and said arithmetic logic operating means, directly makes an access to said host main memory by the address computed by said arithmetic logic operating means, and directly loads the result of said arithmetic logic operating means to said host main memory;

microprogram control means connected to said arithmetic logic operating means and said DMA interface means for controlling those means;

input/output interface means connected to said host central processing unit and said microprogram control means to apply command information received from said host central processing unit to said microprogram control means; and

clock circuit means connected to said arithmetic logic operating means, said microprogram control means and said DMA interface means for generating system clocks to operate those means.

2. A data processing system according to claim 1, wherein the address computation for the host main memory and the computation of operand data taken out from said host main memory are performed together in said arithmetic logic operating means.

3. A data processing system according to claim 1, wherein said clock circuit means includes means for inhibiting the generation of system clocks in order to

produce the system clock in synchronism with the memory clock cycle of said host main memory.

4. A data processing system according to claim 1, wherein said microprogram control means includes a set-up routine and a microprogram memory for storing the arithmetic logic routine, whereby executing the set-up routine, a command information supplied from said host central processing unit is fetched to branch the program to an operation routine designated.

5. A data processing system according to claim 1, wherein said DMA interface means comprises:

a memory address register for holding an address to fetch the data from said host main memory or an address to store data into said host main memory;

a memory data register for holding the result of computation into said host main memory;

a memory storage register for holding the data fetched from said host main memory; and

a DMA controller for holding a memory request signal and a read/write signal for said host main memory, and for receiving an acknowledge signal from said host main memory.

6. A data processing system according to claim 1, wherein said microprogram control means comprises:

a read only memory for storing a microprogram;

a microprogram sequencer for supplying an address in which the microprogram is stored to said read only memory;

a pipeline register for holding each instruction of the microprogram read out from said read only memory;

a branch condition circuit for supplying control signals to said microprogram sequencer on the basis of the conditions produced from said arithmetic control means.

7. A data processing system according to claim 6, wherein said branch condition circuit comprises:

a condition code selector which receives condition signals outputted from said input/output interface means and flag signals outputted from said arithmetic logic operating means, and selectively outputs those on the basis of control signals supplied from said microprogram control means; and

a logic circuit connecting to the output of said condition code selector which receives a condition signal or a flag signal outputted from said condition code selector and control signal from said microprogram control means thereby to supply a branch specifying signal to said microprogram control means.

8. A data processing system according to claim 1, wherein said clock circuit means comprises:

an oscillator for oscillating basic clock signals;

first and second frequency-dividing flip-flop circuits which receive the basic clock signals from said oscillator and produce first and second frequency-divided clock signals;

a logic multiplying circuit for receiving frequency-divided clock signals from said first and second flip-flops to produce a given timing pulse signal;

a disable flip-flop which receives an output signal from said logic multiplying circuit to reset said first and second frequency-divided flip-flops thereby to halt said frequency divided clock signals;

a latch circuit which receives a memory access acknowledge signal from said host main memory to produce a logical high level signal; and

a synchronous flip-flop circuit which receives an output signal from said latch circuit to supply a synchronous signal to said disable flip-flop and to reset said disable flip-flop thereby to make active said first and second frequency-divided flip-flops.

0021365

FIG. 1

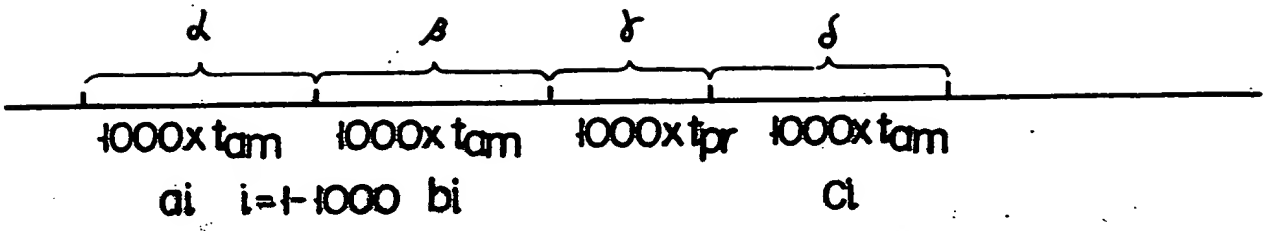


FIG. 3

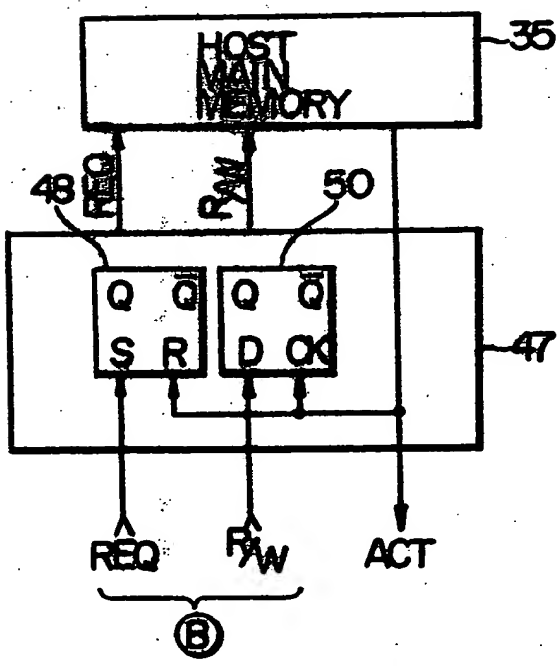
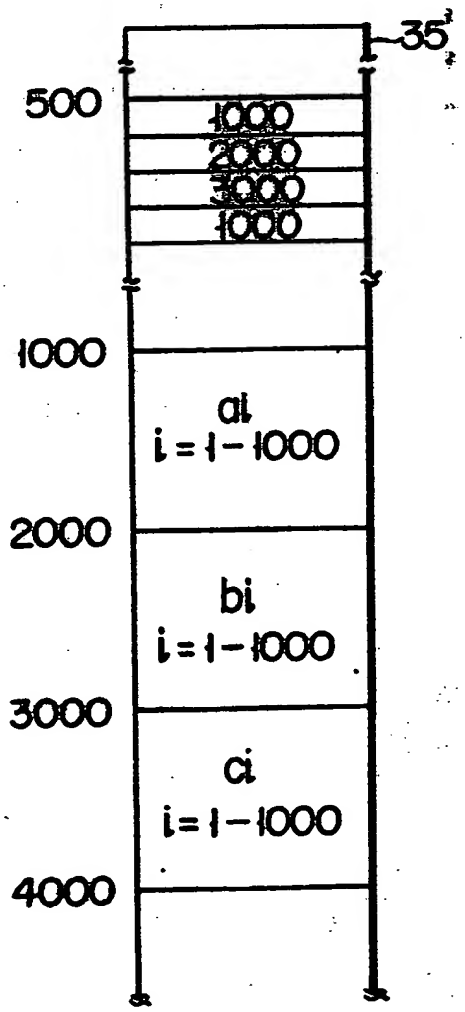


FIG. 4



0021365

FIG. 2

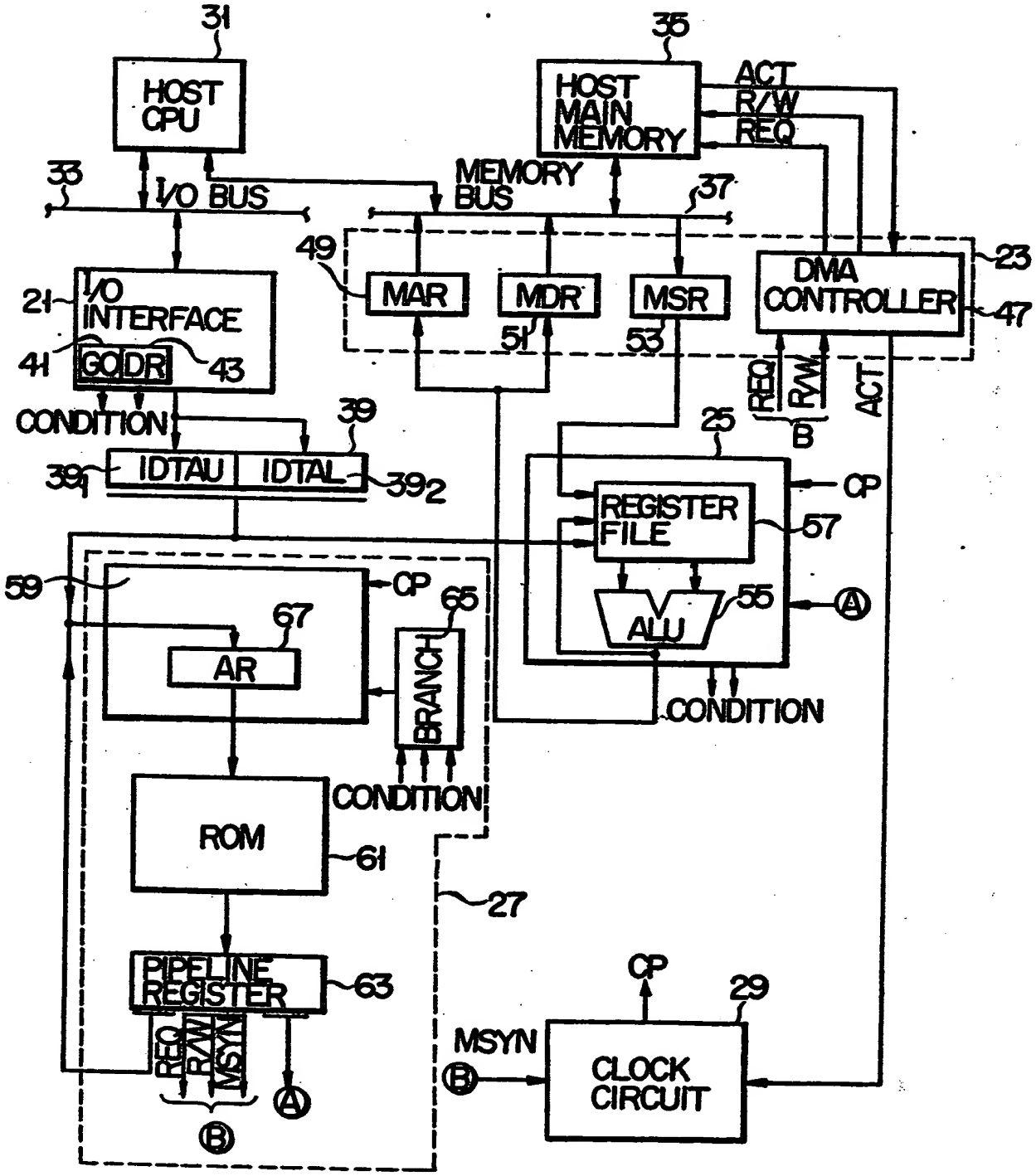
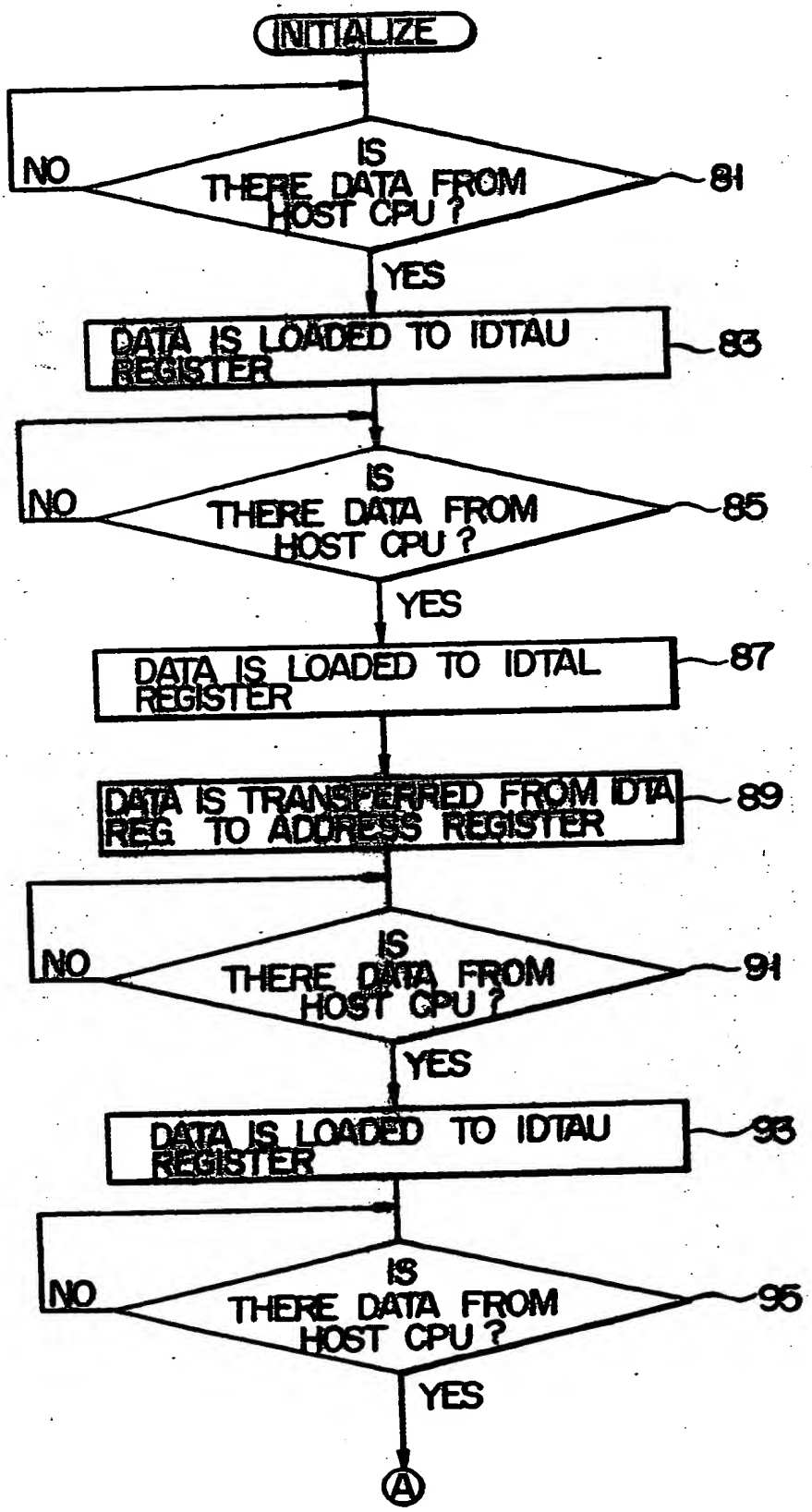


FIG. 5A


INITIALIZE
IS THERE DATA FROM HOST CPU?
NO
YES
81
DATA IS LOADED TO IDTAU REGISTER
83
IS THERE DATA FROM HOST CPU?
NO
YES
85
DATA IS LOADED TO IDTAL REGISTER
87
DATA IS TRANSFERRED FROM IDTA REG TO ADDRESS REGISTER
89
IS THERE DATA FROM HOST CPU?
NO
YES
91
DATA IS LOADED TO IDTAU REGISTER
93
IS THERE DATA FROM HOST CPU?
NO
YES
95
A

0021365

FIG. 6

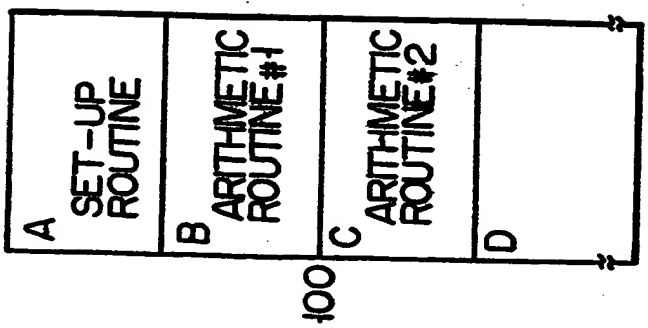


FIG. 5B

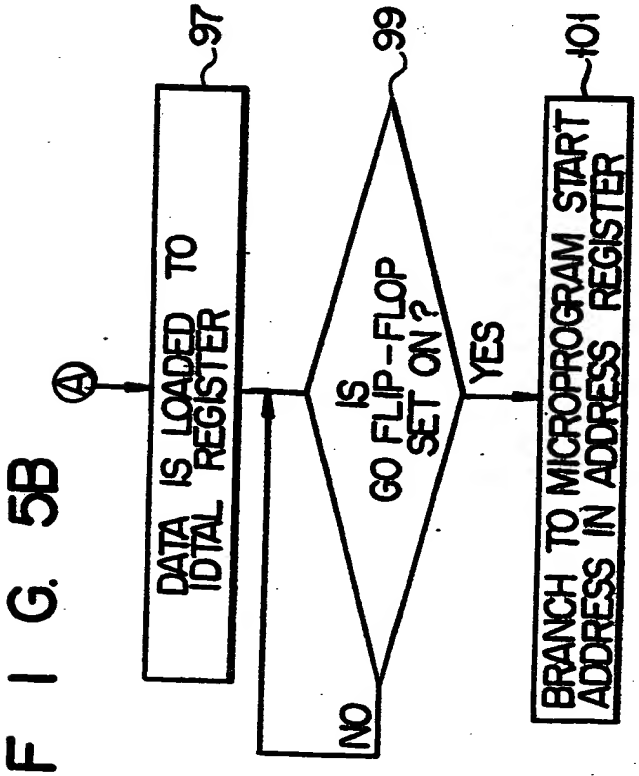
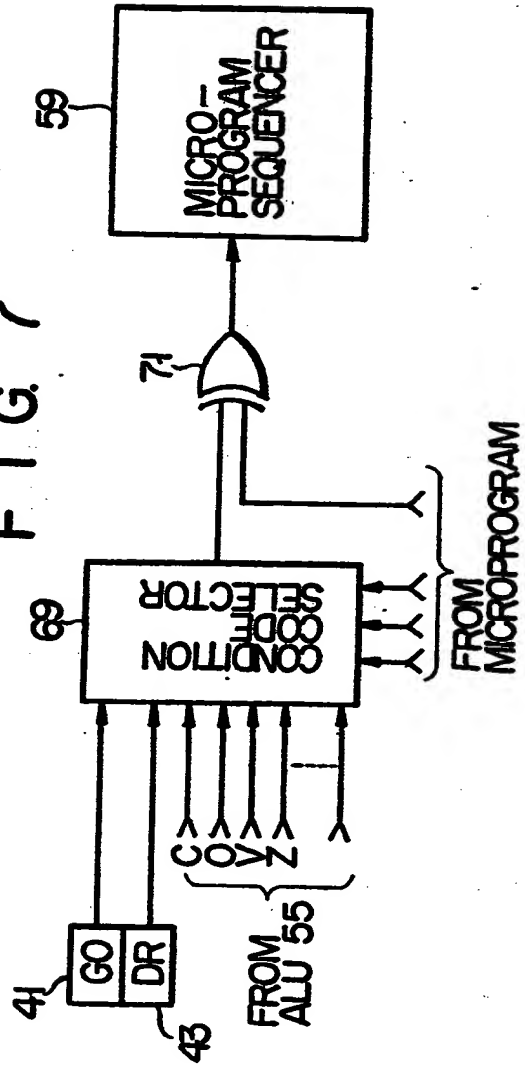
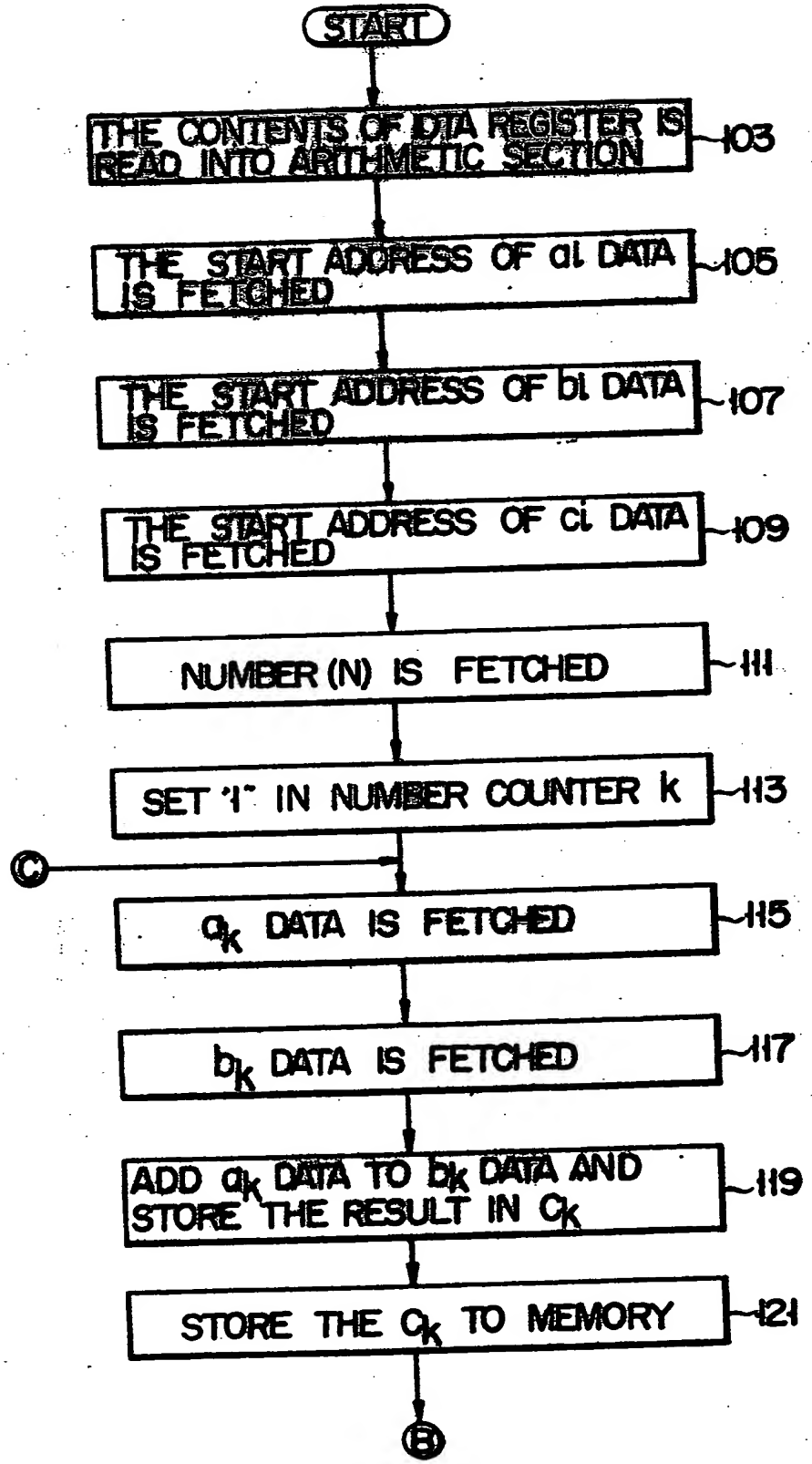


FIG. 7



0021365

FIG. 8A



0021365

FIG. 8B

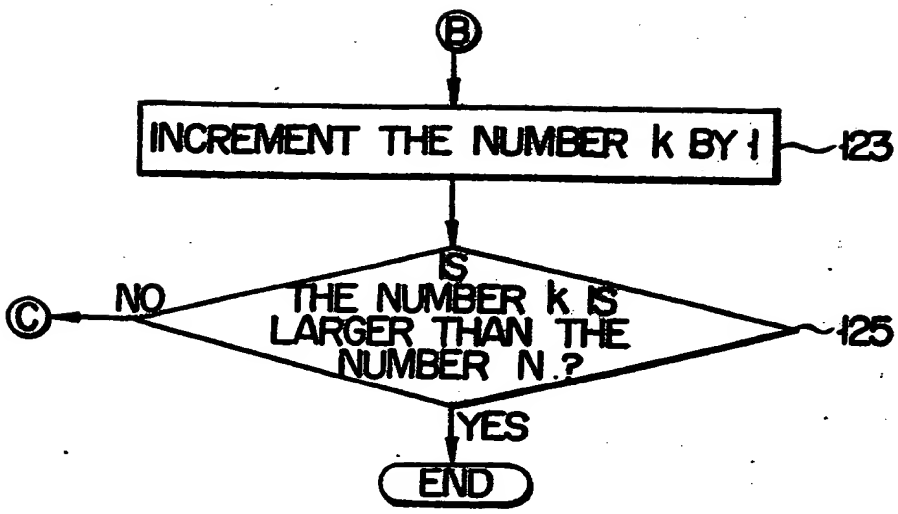
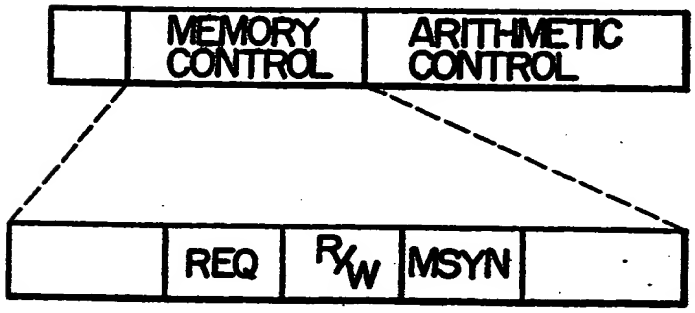


FIG. 9



0021365

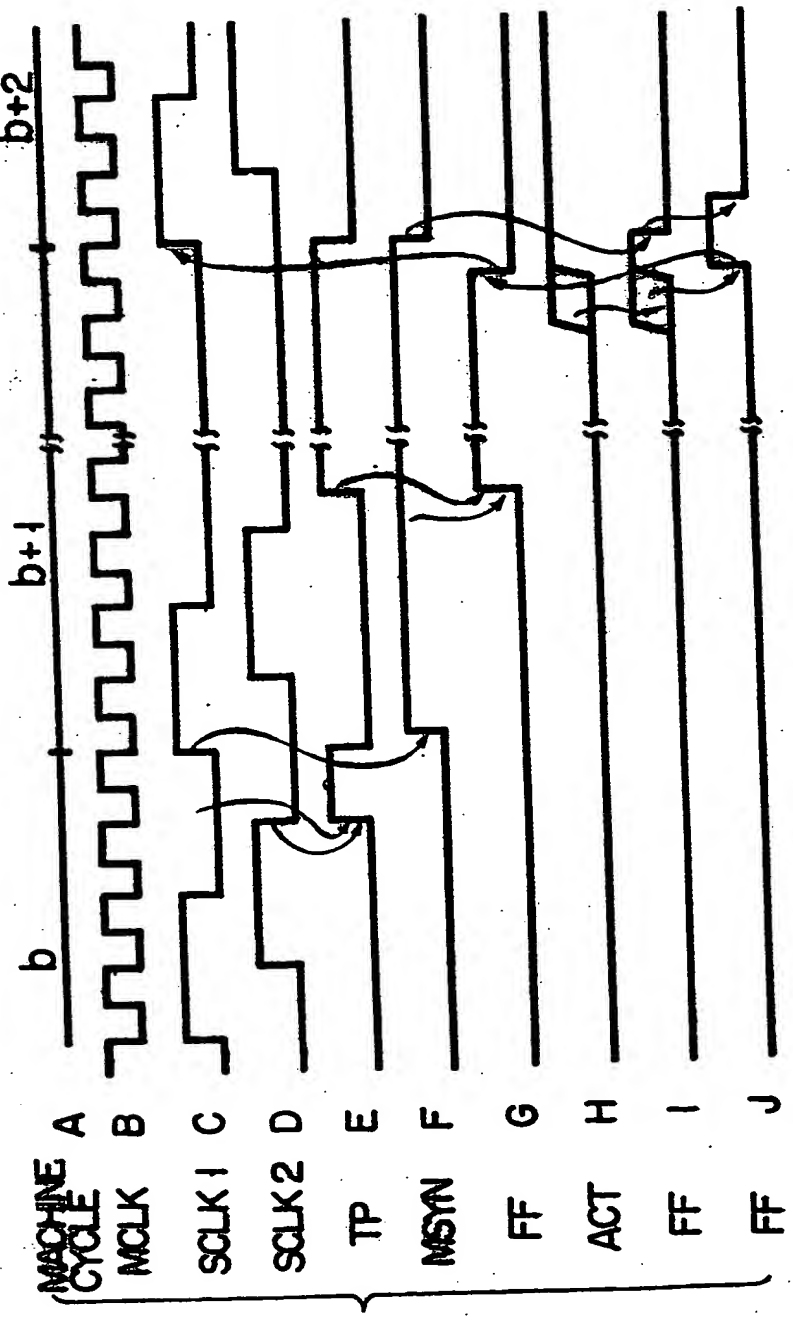


FIG. 10

FIG. 11

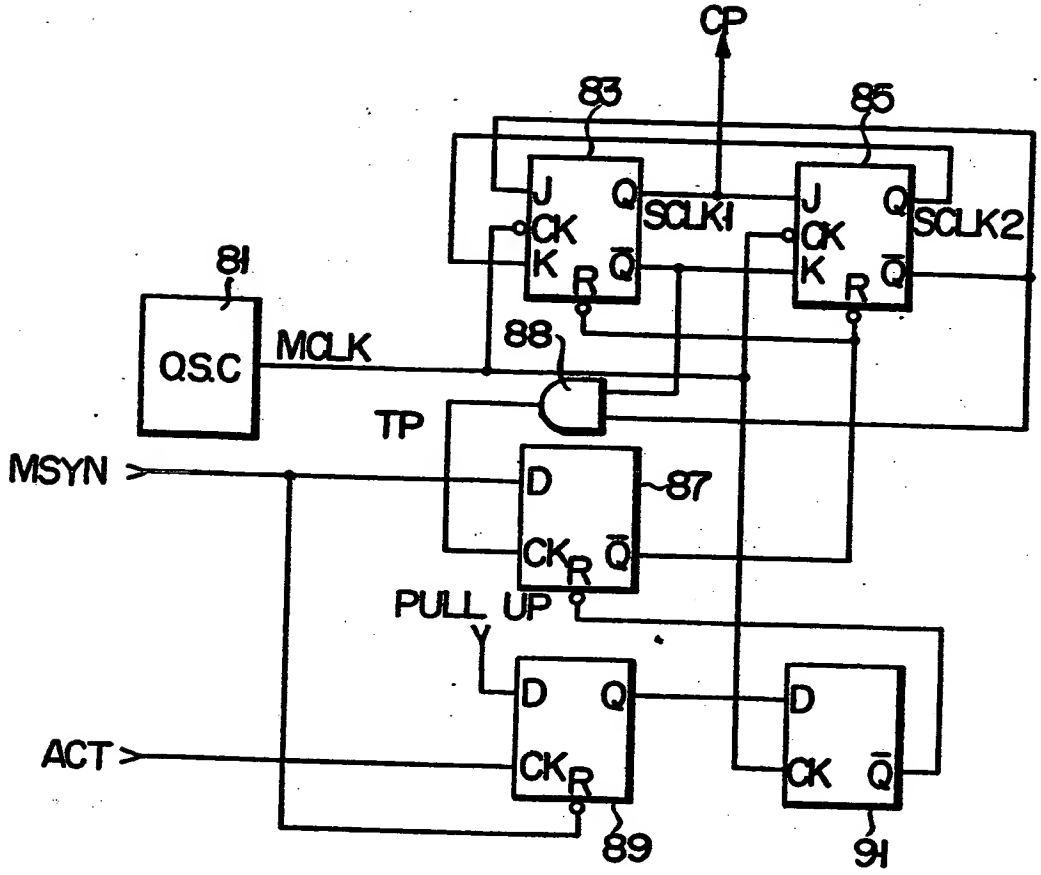


FIG. 12

ADDRESS	MNEMONIC
b	M. REQ
b + 1	M. SYNC
b + 2	NOP
b + 3	NOP
b + 4	NOP
b + 5	NOP

MEMORY CYCLE

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 021 365
A3

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 80103457.0

(22) Date of filing: 20.06.80

(51) Int. Cl.³: G 06 F 15/16
G 06 F 9/38, G 06 F 15/20

(30) Priority: 26.06.79 JP 79662/79

(43) Date of publication of application:
07.01.81 Bulletin 81/1

(88) Date of deferred publication of search report: 05.08.81

(84) Designated Contracting States:
DE FR GB NL

(71) Applicant: TOKYO SHIBAURA DENKI KABUSHIKI KAISHA
72, Horikawa-cho Saiwai-ku
Kawasaki-shi Kanagawa-ken 210(JP)

(72) Inventor: Shibayama, Shigeki
1920 Hiyoshihon-cho Kohoku-ku
Yokohama-shi(JP)

(72) Inventor: Iwata, Kazuhide
891-17-9-201, Kami Shirane-cho Asahi-Ku
Yokohama-shi(JP)

(74) Representative: Patentanwälte
Henkel-Kern-Feiler-Hänzel
Möhlstrasse 37
D-8000 München 80(DE)

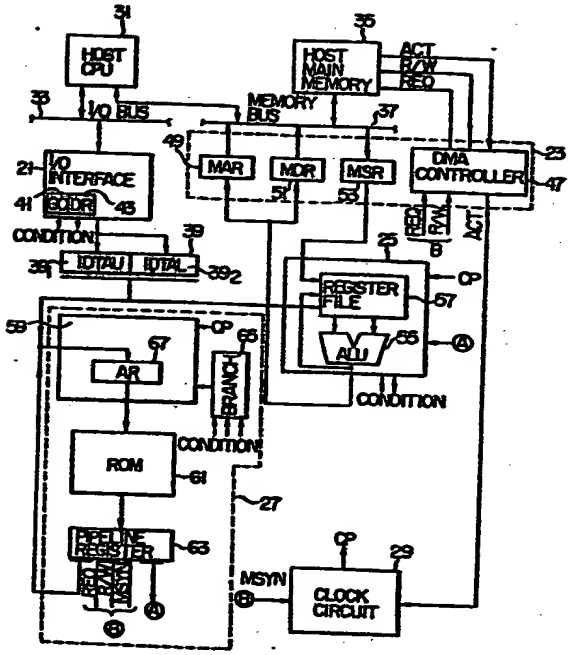
(54) Data processing system with a slave computer.

(57) In a data processing system having a slave computer connecting to a host central processing unit (31) and a host main memory (35), the slave computer includes an arithmetic logic operating means (25). The arithmetic logic operating means (25) calculates an address of the host main memory (35). A DMA interface (23) directly makes an access to the host main memory (35) to fetch operand data into the slave computer. The arithmetic logic operating means (25) computes the operand data under control of a microprogram control section (27) and directly loads the computed result to the host main memory (35) through the DMA interface (23).

EP 0 021 365 A3

./...

FIG 2



European Patent Office

0021365

Application number
EP 80 10 3457

EUROPEAN SEARCH REPORT

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim
	COMPUTER DESIGN, vol. 11, no. 5, May 1972 CONCORD (US) TEICHER: "Implementation of a Hardware Floating Point Processor" pages 122, 124, 126 and 128 * Page 124, line 41 to page 126, line 1 and page 128, left-hand column, line 21 to right-hand column, line 24 * ---	1,2,5
	US - A - 4 128 876 (IBM) * Column 2, line 41 to column 3, line 14 and column 9, line 53 to column 12, line 27 * ---	1,4,6, 7
	US - A - 3 956 738 (HONEYWELL) * Column 3, line 27 to column 4, line 38 and column 8, line 28 to column 13, line 50 * ---	3,6,8
E/P	FR - A - 2 422 205 (RENAULT) * Page 1, line 39 to page 2, line 17 and page 3, line 10 to page 4, line 29 * ---------	1,2,4, 6,7

CLASSIFICATION OF THE APPLICATION (Int. Cl.³)

G 06 F 15/16
9/38
15/20

TECHNICAL FIELDS SEARCHED (Int. Cl.³)

G 06 F 9/38
15/16

CATEGORY OF CITED DOCUMENTS

X: particularly relevant
A: technological background
O: non-written disclosure
P: intermediate document
T: theory or principle underlying the invention
E: conflicting application
D: document cited in the application
L: citation for other reasons

&: member of the same patent family, corresponding document

X The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
The Hague	22.04.1981	LEPEE

EPO Form 1503.1 06.78